A Model for Joint Software Reviews

Gina Kingston

DSTO-TR-0735

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

A Model for Joint Software Reviews

Gina Kingston

Information Technology Division
Electronics and Surveillance Research Laboratory

DSTO-TR-0735

ABSTRACT

Joint software reviews, involving the developer and the acquirer, play an important role in Defence's acquisition of software-intensive systems. However, academic and commerical work on software reviews has focused on intra-organisational peer reviews and software inspections. This report argues that the principles which have been derived for inspections cannot be blindly applied to joint software reviews.

This paper proposes a model of joint reviews, which draws on software engineering, decision and negotiation theory, and models of inspection. The model suggests that the structure and goals of the review group may significantly affect the outcome of the review. The model has also been used to suggest changes to Defence's software review process and to plan a series of studies on joint software reviews. These studies will provide additional and updated recommendations on how Defence should structure their software reviews for maximum efficiency and effectiveness.

RELEASE LIMITATION

Approved for public release

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

Published by

DSTO Electronics and Surveillance Research Laboratory
PO Box 1500
Salisbury South Australia 5108

Telephone: (08) 8259 5555
Fax: (08) 8259 6567

AR-010-661
October 1998

APPROVED FOR PUBLIC RELEASE

A Model for Joint Software Reviews

Executive Summary

Joint software reviews play an important role in Defence's acquisition of software intensive systems. The study of joint software reviews has attracted little attention compared to that given to the related field of software inspections. In recent years, a model for software reviews has been developed. The model was developed for a particular type of review, software inspections. This paper examines some of the differences between software inspections and joint software reviews and the relevance of the model to joint software reviews.

Several limitations are identified with the current model of software reviews when it is applied to joint software reviews. The model is extended using information from organisational behaviour and several hypotheses are developed. These hypotheses are concerned with the difference between joint software reviews and inspections. They propose that the performance of reviews is adversely affected when participants have differing or conflicting goals, or when participants form sub-groups, such as when participants come from several organisations. These organisational factors: goal differences and the presence of subgroups, are of high relevance to joint software reviews, such as those conducted by Defence, where both occur frequently.

This paper proposes several possible changes to the way in which Defence conducts its reviews and organises its contracts. These suggested changes aim to minimise the impact of the organisational factors and improve the performance of joint software reviews. They include: adding goal-identification and goal-resolution phases to the joint software review processes; incorporating incentive schemes into contracts; and the use of integrated product teams. Some of these suggestions have already been successfully introduced on a few Defence projects. This paper helps to explain the potential benefits of these approaches and provides some justification for their use on other projects.

A series of studies into software reviews is planned under DSTO's Joint Reviews of Software (JORS) task. These studies will explore the hypotheses presented in this paper. Reports on these studies will provide updated and additional recommendations on how Defence's joint software reviews can be tailored for increased efficiency and effectiveness.

Author

Gina Kingston
Information Technology Division

Gina has been employed in the Software Systems Engineering Group of the Information Technology Division of the Defence Science and Technology Organisation (DSTO) since graduating from the University of Tasmania with a BSc with First Class Honours in Mathematics in 1990. Her research interests include process improvement and empirical software engineering. She has conducted research into software costing, the analysis of software, and software acquisition. She is currently undertaking a PhD in Software Reviews through the School of Information Systems at the University of New South Wales and working on the JOint Reviews of Software (JORS) task at DSTO.

Contents

Abbreviations

Abbreviation	Description	Page Introduced
ADF	Australian Defence Force	
ANSI	American National Standards Institute	
DSTO	Defence Science and Technology Organisation	
H1-H7 HT1-HT4 HO1-HO3	Hypothesis • General • Transitive • Overarching	15 and 19-21
IEEE	Institute of Electrical and Electronics Engineers	
IR	Implicit Requirement	7
RM	Risk Management	7

Glossary

Term	Description	Page Introduced
Complementary goals	Goals which can be satisfied simultaneously.	17
Conflicting goals	Sets of goals which cannot be satisfied simultaneously, and which therefore require trade-offs to be made.	17
Defect	*"An instance in which a requirement is not satisfied. Here it must be recognized that a requirement is any agreed upon commitment. It is not only the recognizable external product requirement, but can include internal development requirements..."* [Fagan, 1986].	6
Goal-dependent issue	An issue which is subjective and arises from the goals of the reviewer. IR and RM issues are goal-dependent issues.	6
Inspection	A review who's aim is *"...to detect and identify software element defects. This is a rigorous, formal peer examination..."* [ANSI/IEEE-1028, 1989]	2
Issue	*"An instance where a requirement, either explicit or implicit, might not be satisfied"*. Issues include defects, IR and RM issues.	6
Joint software review	*"A process or meeting involving representatives of both the acquirer and the developer, during which project status, software products, and/or project issues are examined and discussed."* [MIL-STD-498, 1996]	4
Review	*"An evaluation of software element(s) or project status to ascertain discrepancies from planned results and to recommend improvement. This evaluation follows a formal process..."* [ANSI/IEEE-1028, 1989].	2
Partitioned into sub-groups	A group is said to be partitioned into subgroups if and only if there exist two or more clearly distinguishable sub-groups such that each member of the group belongs to one and only one sub-group	15

1. Introduction

Joint software reviews form an important part of the Defence Acquisition Process [MIL-STD-1521B, 1985; MIL-STD-498, 1994; CEPMAN 1, 1996; Gabb, 1997], and with the growing popularity of outsourcing, they are becoming more important in the commercial sector [ISO/IEC 12207, 1995].

Like other forms of review, joint software reviews offer a means to evaluate the product being developed early in the acquisition process. Joint software reviews also enable early evaluation of the development process and evaluation of progress against milestones. Furthermore, joint software reviews offer an opportunity for project plans and expectations to be revised. Despite the use, and the potential benefits of joint software reviews, software-intensive systems are often delivered late, over-budget, and with sub-optimal functionality [Earnshaw, 1994; ADO, 1996; Mosemann II, 1995; Keil, 1995; Heemstra, 1992; Lederer and Prasad, 1995; Canale and Wills, 1995; Walsh, 1994].

Furthermore, anecdotal evidence obtained during interviews with DSTO and ADF personnel suggests that these reviews are considered to be inefficient by many of their participants. The author has conducted both formal and informal interviews with review participants, including those with considerable experience with joint software reviews.

Joint software reviews have been poorly studied. There are no well-defined guidelines on how to conduct joint software reviews. Documentation of joint software reviews, including the military standard on reviews offers only limited guidance [MIL-STD-1521B, 1985]. Some "lessons learned" reports have highlighted perceived problems with joint software reviews, but there are no empirical studies to support their conduct in one manner over another.

Work on other types of software reviews and on software inspections suggests mechanisms which may be appropriate for conducting joint software reviews (e.g. see [Wheeler et al., 1996]). However, there are several differences between the work that has been done in these areas and joint software reviews. Therefore, it cannot be assumed that the results from inspections will explain the performance of all software reviews.

This report provides a discussion of joint software reviews drawing on the software inspection literature and relating it to other forms of software reviews - including joint software reviews. A model of software reviews is then proposed by combining inspection models with information about other software engineering processes, and decision and negotiation theories. Finally, the implications of the theory are discussed along with a series of empirical studies designed to explore the theory.

2. Background

There are many forms of software review - from informal peer reviews to formal inspections and joint software reviews. This section provides an introduction to software reviews by comparing inspections and joint software reviews. Joint software reviews were chosen for study in this report because of their relevance to Defence, and inspections were chosen for comparison because they are perhaps the most widely studied form of review.

Throughout this paper, the term review will be used to encompass inspections, walkthroughs and other forms of review as per the IEEE definition of a **review**: *"An evaluation of software element(s) or project status to ascertain discrepancies from planned results and to recommend improvement. This evaluation follows a formal process..."* [ANSI/IEEE-1028, 1989].

Further information on software reviews can be found in the military and ANSI standards [MIL-STD-1521B, 1985; ANSI/IEEE-1028, 1989]. A comprehensive collection of papers on inspections is [Wheeler et al., 1996], and a detailed comparison of three review techniques - Inspections, Walkthroughs and Milestone Reviews - can be found in [Kim et al., 1995]. Kim describes how these forms of review have different purposes, different numbers of participants and different constraints on the product being reviewed - for example the size of the review product may vary with the type of review.

2.1 Software Inspections

Software inspections were first described by Fagan and are one of the most formal[1] review mechanisms commonly used to evaluate software [Fagan, 1976; Wheeler et al., 1996]. The objectives of an inspection are - *"...to detect and identify software element defects. This is a rigorous, formal peer examination..."* [ANSI/IEEE-1028, 1989]. These objectives will be sufficient to distinguish inspections from other forms of review for the purposes of this paper.

Software inspections have been widely studied and there is significant empirical and anecdotal evidence to support their use[2]. (See [Wheeler et al., 1996] for a collection of papers and bibliography on inspections and Table 1 for references to empirical and anecdotal studies). Several variations of Fagan's inspection process have been proposed (e.g. [Bisant, 1989; Fowler, 1986; Knight and Myers, 1993; Martin and Tsai, 1990; Barnard and Price, 1994]) and, despite the plethora of empirical work, there is no clear-cut "best" method for a given situation. Perhaps one reason for this situation is the lack of theoretical foundations for inspections: although this is starting to change - e.g. [Sauer et al., 1996].

---

[1] Note that in this context the term formal does not mean mathematically rigorous as in formal methods, but rather that the process is well-defined, includes entry and exit checks, and encourages the collection of consistent, historical data.

[2] For example, the cost of detecting and fixing defects using inspections with testing is believed to be several orders of magnitude better than the costs of using testing alone.

Table 1: Empirical and Anecdotal Evidence to Support the use of Inspections

Empirical Evidence	Anecdotal Evidence
[Bisant, 1989]	[Ackerman et al., 1984]
[Brothers, 1992]	[Fagan, 1986]
[Buck, 1981]	[Grady and Van Slack, 1994]
[Knight and Myers, 1993]	[Rifkin and Deimel, 1994]
[Letovsky, 1987]	[Weinberg and Freedman, 1984]
[Myers and Knight, 1992]	
[Porter et al., 1995a]	
[Schneider et al., 1992]	

Guidelines for inspections [Brykczynski, 1994; Gilb, 1996; Grady and Van Slack, 1994; Shirey, 1992] suggest that inspections may fail because of:

1. High start-up costs including cultural change.
2. Poor planning, including introducing inspections on a project which is already in trouble, lack of resources, conducting inspections too late, or rushing inspections.
3. Lack of commitment to the (intent of the) process.
4. Lack of, or poorly defined, inspection goals.
5. Lack of, or differing, standards or quality goals.
6. Inappropriate or untrained reviewers.
7. Lack of entry and exit criteria.
8. Poor product stability.
9. Lack of historical information on defect distribution (insertion and removal by phase and defect type), the cost of inspections and testing, the cost of rework and the cost of defects remaining in the system.

Entry
Planning
Overview
Individual Preparation (Familiarisation or Error Detection
Meeting (Error Detection or Error Collation)
Repair (Fixing mistakes)
Follow-up
Exit

Figure 1: Phases of the Inspection Process

Most of the software inspection methods follow the same basic procedure (see Figure 1), with about 2 hours allocated for each of the individual preparation and the meeting phases [Wheeler et al., 1996]. The most significant differences between inspection processes derive from how the individual preparation phase is spent. In simple terms, if the individual preparation phase is spent on familiarisation with the product, the goals of the review, and the error detection method used, then the meeting phase is spent on error detection [Ackerman et al., 1984; Shirey, 1992]. Conversely, if the individual preparation phase is spent on error detection, then the meeting phase is

spent on error collation [Shirey, 1992; Fagan, 1976; Lanubile and Visaggio, 1995; Tripp et al., 1991; McCarthy et al., 1996]. There is some debate over whether meetings are necessary in these circumstances [Votta, 1993; Sauer et al., 1996; Macdonald et al., 1995].

Other differences in the inspection process occur due to the method used to detect defects. Early methods were ad-hoc or based on checklists [Baldwin, 1992; Fagan, 1986; Fowler, 1986; Shirey, 1992]. More recent techniques are based on decomposition of either the product [Cheng and Jeffery, 1996b], or of the types of defects being detected [Knight and Myers, 1993; Parnas and Weiss, 1985; Porter et al., 1995a].

Different inspection methods also vary: the number of participants [Bisant, 1989; Humphrey, 1995; Porter et al., 1995b; Wheeler et al., 1996]; the participants roles and; whether the process is conducted once, or repeated [Martin and Tsai, 1990; Schneider et al., 1992; VanHee, 1996].

2.2 Joint Software Reviews

Joint Software Reviews differ from software inspections in a number of additional ways and, as they have different purposes, it may be beneficial for both joint software reviews and software inspections to be conducted for the same project. For example, inspections may be conducted before joint software reviews and may provide input to joint software reviews.

According to [MIL-STD-498, 1996] a **joint software review** is *"A process or meeting involving representatives of both the acquirer and the developer, during which project status, software products, and/or project issues are examined and discussed."*

Some of the main differences between inspections and joint software reviews are apparent from this definition. Firstly, the joint review process is, as its name suggests, a joint process. It involves representatives from at least two organisations or groups, namely the acquirer and the developer.

These two groups may have very different functions and may be made up of people with very different backgrounds, experiences, aims and objectives [Gabb et al., 1991; Fisher et al., 1997; Warne and Hart, 1995]. For example, the acquirer's group may include users of the system with little or no background or experience in software engineering.

This is in stark contrast with software inspections, where participants are limited to the product author's peers. These reviewers are usually from the same development team as the product author [Wheeler et al., 1996].

The other main difference is that this definition addresses "project status, software products and/or project issues" being "examined and discussed" rather than defects being identified.

The current procedures for, and work on, joint software reviews are discussed in Section 2.3 before examining the impact of these differences in Section 2.4.

2.3 Joint Software Review Procedures

There are two types of Joint Software Review: technical and management reviews [MIL-STD-498, 1996; ISO/IEC 12207, 1995]. Management reviews occur after technical reviews, and are focused on the cost and schedule, as issues which could not be resolved at the technical review [MIL-STD-498, 1994]. Technical reviews are the focus of this paper, because technical reviews are where most issues are raised, and because technical reviews are more like software inspections. Unless otherwise stated, the term joint software review will be used to refer solely to technical reviews for the remainder of this paper.

The purpose of joint technical software reviews varies considerably from that of Software Inspections. According to [MIL-STD-498, 1994], the objectives of Joint Technical Reviews (software) are to:

"*a. Review evolving software products... review and demonstrate proposed technical solutions; provide insight and obtain feedback on technical effort; (bring to the) surface and resolve technical issues.*

b. Review project status; (bring to the) surface near- and long- term risks...

c. Arrive at agreed-upon risk-mitigation strategies...

d. Identify risks and issues to be raised at joint management reviews.

e. Ensure on-going communication..."

Thus, the purpose of joint software reviews is for both the acquirer and the developer to raise and resolve issues: technical, managerial and risk. Issues which cannot, and usually should not, be resolved at this level are referred to other authorities.

Very little has been documented on joint software reviews. There are few guidelines on how they should be conducted, little discussion on the roles of participants and no debate on how issues should be identified, raised or resolved. Most of the documentation which does exist comes from the military domain [MIL-STD-1521B, 1985; DOD-STD-2167A, 1988; MIL-STD-498, 1994] and these standards may conflict when used without tailoring [Gabb et al., 1992]. Guidelines within these standards are general in nature (e.g. recommending that minutes be taken and that the meeting be co-chaired) or guidelines on what documents and activities should be assessed at the same review. There is little evidence - either theoretical or empirical - to support any of the existing guidelines.

Consequently, the joint software review process is often ad-hoc. The author conducted interviews with several Defence and DSTO personnel during January 1997, which indicated that:

- the material under review may or may not be received prior to the meeting, it may or may not be complete, and the time available to review the documents may vary from a few days to a few weeks;
- prior preparation - either familiarisation or issue identification - may or may not be conducted;
- people who review material prior to the meeting may either forward their comments or attend the meeting;

- people who review material prior to the meeting may or may not be given guidance as to the types of issues they are to try and identify;
- reviewers are not given guidance or training on how to review material but may use criteria which they have devised;
- there are more participants at joint software reviews than at typical inspections (for example at one review there were 7 participants from each of Defence and the contractor, as well as approximately 13 observers);
- some participants are not happy with review meetings as the meetings are perceived to drift from their main purpose;
- participants may include outside experts in technical areas, quality management and ordnance;
- usually the reviews are held at the contractor's premises and the contractor writes the agenda which starts with actions outstanding;
- the reviews may include presentations and demonstrations as well as a discussion of any documentation;
- participants may not have an active role in the meeting other than identifying issues.

2.4 Issues and Defects

Software inspections tend to look at defects whereas joint software reviews tend to look at issues. This report explores the differences between these terms. The terms are first defined and examples of issues and defects are given. This section then identifies the similarities and differences between defects and issues and looks at the types of checklists used to uncover them.

Defects
Fagan defines a **defect** as *"an instance in which a requirement is not satisfied. Here it must be recognized that a requirement is any agreed upon commitment. It is not only the recognizable external product requirement, but can include internal development requirements..."* [Fagan, 1986].

Although several alternative definitions of the term "defect" have been proposed Fagan's definition is used because: 1) it comes from the field of software inspections and is consistent with the term's normal usage within that field, and 2) it provides a workable definition within the context of this document.

Issues
For the purposes of this work, an **issue** is defined as *"an instance where a requirement, either explicit or implicit, might not be satisfied"*. This definition, together with Fagan's definition of a defect, highlights the difference between issues and defects. Defects are a sub-class of issues - a defect is an issue where it is known that a requirement is not met. This paper also identifies two other types of issues: those related to risk mitigation (**RM** issues), and those related to implicit requirements (**IR** issues). RM and IR issues are both termed goal-dependent issues as their evaluation depends on the goals of the reviewer. Figure 2 shows the relationship between issues, defects, RM issues and IR issues.

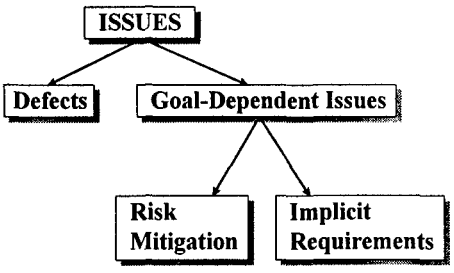

Figure 2: Classification of Issues

Gabb argues that IR issues are to be expected as part of joint software reviews: *"In many projects the refinement of requirements and detailed design can lead to implementations that the customer regards as unsatisfactory. This is particularly likely in areas such as the definition of the user interface and in the specification of detailed performance (such as response times). More importantly, although the implementation may be unacceptable, it is often either compliant with the higher level requirements or the judgment of compliance is a subjective issue. While it might be claimed that this is the result of poorly specified requirements, this will frequently not be the case. Customers are encouraged to avoid detail in the requirements which might inhibit the design... The penalty for lack of detail is a development resulting in an unacceptable design."* [Gabb et al., 1992]

Example 1 provides an extreme example of how risk mitigation issues may also arise. In practice, hardware controls and additional quality assurance measures would be in place to reduce the risk. However, the risk would still exist at the system level.

Example 1: Power Plant
A software system is being developed to control a nuclear power plant. One of the requirements for this system is that the system must be able to implement measures to prevent a nuclear meltdown within a (given) short time of a problem being detected.

When the design for this system is under review, it may not be possible to determine absolutely if the timing requirements for this system will be met. Different designs will have different strengths and weaknesses in this area and some may make it easier to show that this requirement is likely to be met. This becomes an issue if someone believes (based on his or her experience or otherwise) that the risk of the requirement not being met is unacceptable.

Although this paper is the first to distinguish the three types of issues - defects, IR and RM issues - questions or criteria aimed at identifying each type of issue can be found in existing checklists for reviews and evaluations (as in Figure 3 to Figure 6).

Logic
Missing
1. Are all constants defined?
2. Are all unique values explicitly tested on input parameters?
3. Are values stored after they are calculated?
4. Are all defaults checked explicitly tested on input parameters?
5. If character strings are created are they complete, are all delimiters shown?

...

Figure 3: A Sample Design Inspection Checklist [Fagan, 1976]

Inspections have focused on the detection of defects, so not surprisingly the questions in inspections checklists are aimed at detecting defects. Parts of two such checklists are shown in Figure 3 and Figure 4. The questions on these checklists are often worded so that they have yes/no answers. (Although even with inspections, issues may be identified which are false positives[3] rather than defects e.g. [Votta, 1993; Sauer et al., 1996].)

Completeness
1. Are all sources of input identified?
2. What is the total input space?
3. Are there any "don't care" sets in the input space?
4. Is there a need for robustness across the entire input space?
5. Are all types of output identified?

...
Ambiguity...
Consistency...

Figure 4: A Sample Requirements Checklist [Ackerman et al., 1989]

There has been very little formal study of other types of review, in particular there has been little study of reviews that identify issues such as the RM and IR issues identified in this paper. Nevertheless, criteria aimed at identifying RM issues can be found in checklists for joint software reviews and checklists for other forms of product evaluation. Examples include Feasibility (Figure 6, point h), Testability (Figure 6, point i), Maintainability, system cost and completion dates.

Guidelines on criteria for identifying RM issues can be formulated by considering the examples in Figure 5 and Figure 6 together with the nature of risk. The identification of risk is often subjective [Tversky and Kahneman, 1989; Haimes, 1993]. It is therefore unlikely that specific questions with yes/no answers can be used to identify risks. In the previous examples, all the criteria aimed at identifying RM issues are stated in terms of the area of interest rather than as pointed questions. These criteria will usually address the future state of the system's products and properties. In software engineering, risk is most commonly defined as a combination of the consequences of an undesirable event, and the likelihood ($0<likelihood<1$) of its occurrence [Charette, 1994; Rescher, 1983]. So once the event has occurred (or it is certain that an event will occur) it is no longer considered a risk (its likelihood is 1) and the event becomes a

[3] False positives are issues which the review believed were defects, which are actually correct.

problem or defect. Thus, RM issues must relate to the future state of the system's products and properties.

Functionality
5.1.1 *Suitability*: The capability of the software to provide an appropriate set of functions for the specified tasks and user objectives
5.1.2 *Accuracy*: The capability of the software to provide the right or agreed results or effects.
5.1.3 *Interoperability*: The capability of the software to interact with one or more specified systems.

...

Figure 5: Quality Characteristics [ISO/IEC 9126-1, 1996]

Despite the lack of prior study and classification of IR issues, criteria aimed at identifying IR issues can also be found in checklists for joint software reviews and in checklists for other forms of product evaluation. For example: Suitability (Figure 4, point 5.1.1), Covering the Operational Concept (Figure 6, point g) and Useability are often poorly or only partially defined and so lead to the detection of IR issues. Because IR issues relate to implicit requirements, they may also be subjective. Therefore, they are generally stated in a similar manner to those of RM issues, which may also be subjective. Issues related to implicit requirements are concerned with the system from the perspective of its entire user population (including support personnel) or with the system from an operational perspective.

Note that defects may also be detected by broad questions such as those used to detect RM and IR issues. For example, Porter's ad-hoc technique uses a list of criteria similar to those in Figure 5 [Porter et al., 1995a]. Also, the criteria which compare the current and previous system products and properties (as in Figure 3 and Figure 5-point a), often result in the detection of defects and criteria which address the completeness and internal and external consistency of a product may result in the detection of defects. For example, criteria a) and e) in Figure 6 are similar to the criteria given in Figure 4 (although specific questions are given in Figure 4 and not in Figure 6). Clearly defects can be identified using both checklists.

Requirements
a) Contains all applicable information from the Software System Specification, the Interface Requirements Specification (IRS) and the relevant Data Item Descriptions (DIDs).
b) Meets the Statement of Work (SOW), if applicable
c) Meets the Contract Data Requirements List (CDRL), if applicable
d) Understandable
e) Internally consistent
f) Follow Software Development Plan
g) Covers the Operational Concept
h) Feasible
i) Testable

Figure 6: Evaluation Criteria [MIL-STD-498, 1994]

Figure 7 highlights the differences between defects, IR issues and RM issues. Defects can normally be objectively identified [Remus, 1984] while, as stated previously, the identification and resolution of RM issues and IR issues may be subjective [Gabb et al., 1991; Gabb et al., 1992; Fisher et al., 1997]. IR issues and defects are also assessed based on the current state of the system, while RM issues refer to future states of the system[4].

	Objective	**Subjective**
Now	Defect	Implicit Requirement
Future		Risk Mitigation

Figure 7: Characteristics of Different Classes of Issue

The identification and resolution of RM issues are subjective. One reason is differences between the reviewers' perceptions of risk. Their perceptions are influenced by a number of factors including framing [Tversky and Kahneman, 1989], background, experience and the goals of the reviewer.

IR issues can depend on the goals of the reviewer with respect to the relative importance of the system's operational requirements, user interface requirements, and desirable, but conflicting quality and performance requirements [Gabb et al., 1991; Ulkucu, 1989]. For example, *"There appears to be an almost universal difference of opinion between developers and customers regarding the suitability of delivered documentation"* [Gabb et al., 1991].

Because of the dependence of RM and IR issues on the reviewers' goals, they are collectively referred to as **goal-dependent issues** in this paper.

In practice the resolution of goal-dependent issues tends to involve a process of negotiation. The acquirers and developers first identify issues either independently or jointly. For each issue identified they then discuss how the issue should be addressed: what action is required, whether it has already been addressed, or whether no further action is required. The subjective nature of these issues means that considerable time may be spent on these negotiations [Gabb et al., 1992].

[4] Since the future state of the system cannot be objectively determined, RM issues are subjective and the only type of issues which refers to the future state of the system.

3. Models

A model of factors affecting the performance of software inspections has recently been proposed [Sauer et al., 1996]. Using this performance model and results from behavioural theory, Sauer et al argue for the model of software inspections shown in Figure 8. Defects are identified by individuals, collated and if necessary, assessed by a pair of experts (an expert-pair). This assessment is aimed at discriminating between true defects and false positives - things which a reviewer claimed were defects, but which should not have been identified as defects [Porter et al., 1995b; Sauer et al., 1996; Votta, 1993]. This assessment is not performed for (claimed) defects that were identified by more than one reviewer - such defects are assumed to be true defects and not false positives.

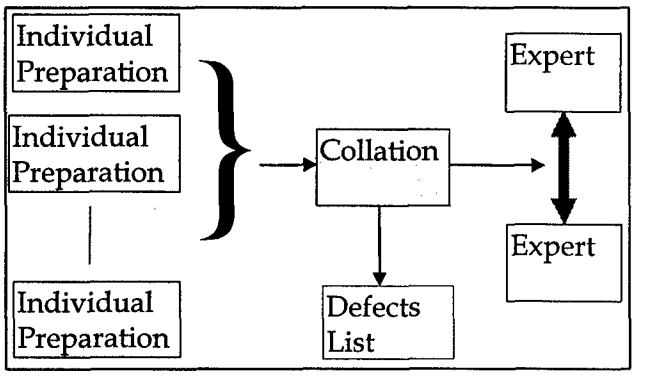

Figure 8: An Alternative Inspection Process [Sauer et al., 1996]

This paper argues that Sauer's model of inspections cannot be used for joint software reviews because these reviews have participants from multiple organisations and consider goal-dependent issues. Sauer's model of inspections uses identification by multiple reviewers or an expert pair to discriminate between true defects and false positives. Example 2 shows how a goal-dependent issue identified in a joint software review may not require any action even when the issue is identified by multiple reviewers. As previously discussed, the subjective nature of issues [Gabb et al., 1991] means that they are typically resolved by negotiation during the review.

Example 2: Language Choice
Consider the joint review of a software development plan where the developers wish to use the C programming language and the acquirers believe that the Ada programming language should be used. (Based on the author's experiences, similar situations are not uncommon.)

Two acquirers identify the choice of development language as an issue. They believe that Ada should be used because it is a defacto defence standard [DI(N)LOG 10-3, 1995], and they believe there would be less risk in the development of the system because of the rigorous development procedures it encourages [Marciniak, 1994].

Without discussion with the developers, it may appear that the development plans should be amended and Ada used as the development language. However, during the review, the developers describe how they have considerable expertise in developing similar systems in C and how they believe that most of the system could be developed using reusable components

from previous systems [Anderson et al., 1988]. Based on this additional information, both the acquirers and the developers agree that the system should be developed using the C programming language. However, as part of their risk mitigation strategy, the acquirers want to review this decision when more information is available about the level of reuse that could be achieved.

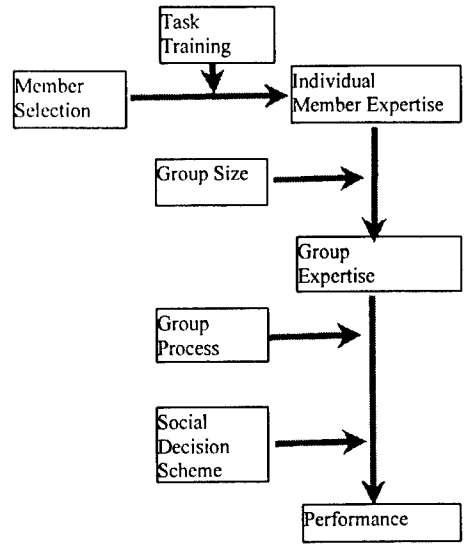

Figure 9: A model for the performance of software inspections [Sauer et al., 1996]

Thus, this paper proposes a new model for the performance of software reviews which combines the model of the performance of software inspections[Sauer et al., 1996] with a model of negotiation [Davis and Newstrom, 1985].[5] Sauer et al's model is shown in Figure 9, Davis and Newstrom's model is shown in Figure 10, and the new model is depicted in Figure 11. The new model is currently being refined into an integrated model, which more clearly shows the relationship between the factors and the structure of the review group. The new model will be published in the author's PhD thesis. However, the factors in the current model are similar to those being proposed in the new model, and it is sufficient to explain some of the differences between joint reviews and inspections.

[5] Sauer et al's alternative inspection process, Figure 8, was derived from their model of the factors effecting the performance of software inspection, Figure 9.

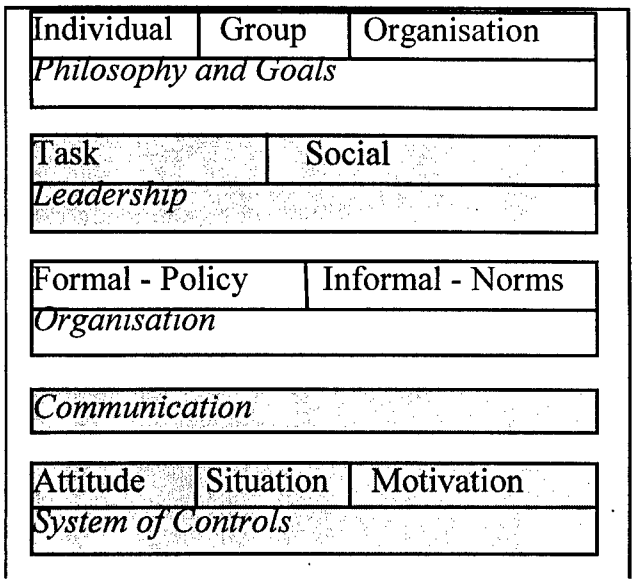

Figure 10: A model of factors effecting the performance of organisations [Davis and Newstrom, 1985].

The boxes all three models show factors that can effect the performance of software reviews. The factors from Davis and Newstrom's model appears in the lower left of the new model. In the new model it is assumed that they influence, and are influenced by the same factors. The lines in the Sauer et al's model, and in the new model show influence. So for example, group experience influences performance and individual expertise influences group expertise. Furthermore, the way individual expertise influences group expertise is modified by the group size. This is shown by the arrow from group size to the line joining individual expertise and group expertise.

The box in the upper left hand corner of the new model does not appear in either of the other models. The ovals in this box indicate different sources of people from which the participants can be drawn. The potential pool of participants can influence how members are selected. For example, if there are few potential participants they may all be selected, while if there are many potential pariticipants they may be selected on the basis of availability or expertise. The member selection determines the individuals who participate in the review and thus influences the individual expertise. The selection of members also influences the properties of the group such as the leadership and organisation of the group. The selection of members from different organisations can effect the philosophy and goals of the organisations present at the review. The selection of different individuals may also effect the communication of the group: either through different styles of personal communication, or though access to different communications media, such as email and groupware.

The proposed model differs from the performance model of Sauer et al by making explicit the dependence of the performance of software reviews on: 1) the goals of the participants, leadership, organisation, communication and system of controls; and 2)

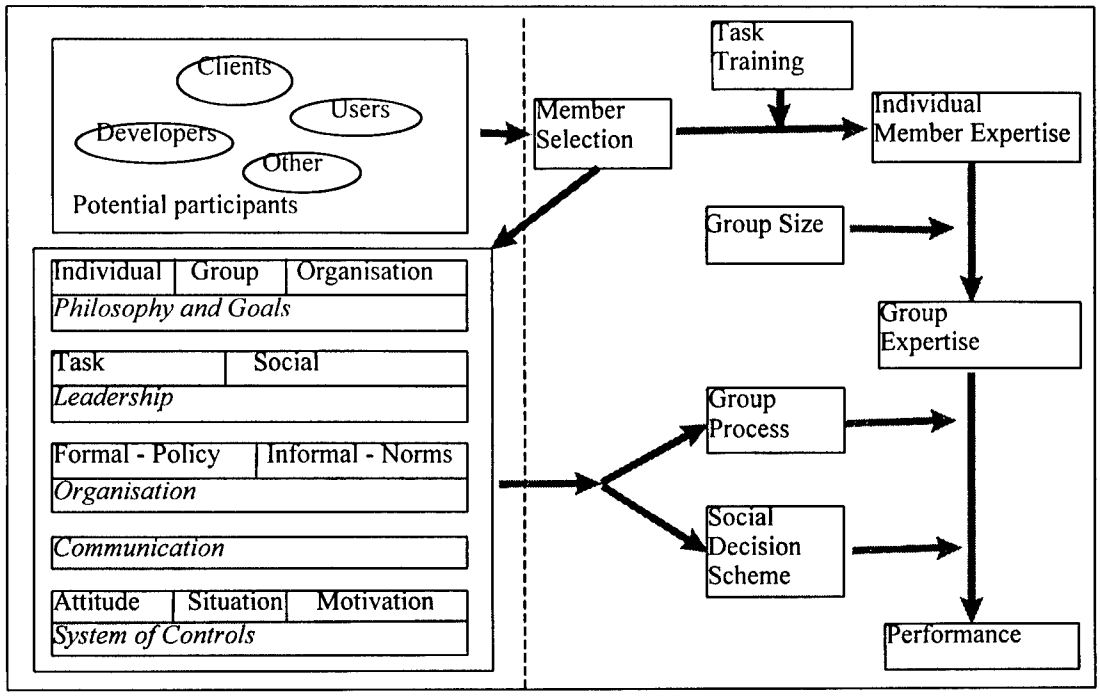

Figure 11: Group Performance [Sauer et al., 1996; Davis and Newstrom, 1985]

the different expertise and backgrounds of the participants. Section 3.1 discusses the application of the model to software reviews and identifies two areas that deserve further attention. These areas: group structure and goal conflict are discussed later in this section.

3.1 Application of the model to software reviews.

The performance model used by Sauer is based on a behavioural task model [Sauer et al., 1996]. The representation of the behavioural task model is identical to the performance model used by Sauer. The behavioural model looks at tasks such as ranking items according to their potential benefits. The main difference between software reviews, inspections and the behavioural tasks are the number of items that must be evaluated, and the purpose of the evaluation. In software inspections, the number of defects is not known, and each potential defect is either correct or incorrect. In the behavioural model, the number of items to be ranked is known, and the purpose is to judge the relative merits of all the items. In joint software reviews, the number of issues is not known, and potential issues may be evaluated relative to the goals of all participants. Issues must be consistently evaluated and can be combined during the group phase. Figure 12 is an extension of Sauer et al's comparison between inspections (which he calls Software Development Technical Reviews) and the behavioural task model. It incorporates a column on generic software reviews that allow the identification of goal-dependent issues. These generic software reviews include joint software reviews. From Figure 12 it can be seen that generic software reviews may be a

better match than software inspections to the characteristics of the behavioural task model, partly because the group activity is no longer one of validation or binary discrimination.

	Inspections	Behavioural	Software Reviews
Task stages	2	2	2 (varies)
Task items	defects	items	issues
Multiple items, #	unknown	known	unknown
Individual activity	discovery	judgment	discovery & judgment
Group activity	validation - binary	judgment	judgment
Individuals can be undecided about an item	yes	no	yes

Figure 12: A comparison between inspections, behavioural task models [Sauer et al., 1996] and a task model for software reviews which may involve issue identification and resolution.

The remaining components of the model come from a generic model of group meetings [Davis and Newstrom, 1985]. While this model has not previously been used for software reviews, its use is supported by results from negotiation and decision support theory (e.g. [Foroughi et al., 1995; Nielsen, 1979; Nunamaker et al., 1991a]), organisational behaviour (e.g. [Luthans, 1985]), and the performance of software programmers (e.g. [Chung and Guinan, 1994; Simmons, 1991]). Furthermore, many of these areas have been identified as contributing to the performance of software reviews [Parnas and Weiss, 1987; Gilb, 1996; Brykczynski, 1994; Weller, 1993].

Of these areas, there are two areas for which there are considerable differences between joint software reviews and inspections. These areas are group structure and goal conflict. The remainder for this report focuses on these two areas. Section 3.2 discusses group structure. Section 3.4 discusses goal conflict and is preceded by a discussion of how goal conflict arises in software reviews.

3.2 Group Structure

The new model for software reviews (Figure 11) suggests that the structure of the review group will affect the performance of software reviews. Three components of the model support this suggestion: the potential participants pool, the method of selecting members, and the organisation of the group. The pool of potential participants and the method of selecting members are particularly important where participants come from multiple organisations. The impact of having the group partitioned into sub-groups within the review group (e.g. participants from two organisations) is the focus of this section.

Work on decision support systems has compared problem-solving sessions (negotiations) between two parties with problem solving sessions within a single group. Negotiations follow a different pattern, and are generally longer, than problem solving sessions within a single group [Luthans, 1985].

Although the impact of sub-groups has not been investigated for software reviews or inspections, other areas of group organisation have been. Sauer proposes a model to

explain when actual groups perform better at software inspections than nominal groups[6] [Sauer et al., 1996]. Sauer's model is based on the literature from behavioural theory, which states that social decisions are based on plurality effects. That is, a group decision is made once a majority agrees or, where there is no majority, a group decision is made when two or more members agree on a "correct or accurate solution". However, with joint software reviews, there is often disagreement on what constitutes a "correct or accurate solution" and different members often support opposing views [Gabb et al., 1991].

Groups which decompose the inspection task (so that different reviewers address different questions or different parts of the document) have been found to perform better than other groups [Knight and Myers, 1993; Porter et al., 1995a; Parnas and Weiss, 1987]. Sauer suggests two reasons for the improved performance of groups employing these strategies: the first is that the size of the activities has decreased, and the second is that different types of expertise have been made available [Sauer et al., 1996].

Based on work from organisational behaviour [Luthans, 1985], for the purposes of this paper a group is said to be *partitioned into sub-groups* if and only if there exist two or more clearly distinguishable sub-groups such that each member of the group belongs to one and only one sub-group. For example, a group may be partitioned into subgroups if the participants come from two different organisations.

One approach to determining if a group really is partitioned into subgroups, would be to check the cohesion of the group and the cohesion of the sub-groups. The cohesion of the subgroups is expected to be higher than the cohesion of the group under most circumstances [Luthans, 1985].

Work from organisational behaviour suggests that a group which is partitioned into sub-groups would not be as efficient as other groups [Luthans, 1985]. This leads to the following hypothesis for software reviews:

HO1. Software reviews *without partitioned subgroups* will raise and resolve a *greater* number and quality of issues than groups with partitioned subgroups.

Care needs to be taken in evaluating this hypothesis as there are several compounding factors that could occur. Many methods of partitioning a group into subgroups will also decrease the activity required by each subgroup or increase the effective expertise of the group. In these circumstances, the performance of reviews may increase [Sauer et al., 1996]. These compounding effects on review performance may outweigh any performance loss due to the presence of partitioned subgroups [Sauer et al., 1996].

3.3 Goal Conflict within Software Reviews

The presence of multiple groups in software reviews is often associated with goal conflict. Goal conflict in software reviews can stem from two sources. The first source

[6] A nominal group is a collection of individuals that are treated as if they were a group even though there is no group interaction.

is conflict between the goals of the various participants. This source of goal conflict is particularly common where two organisations are present in the review. For example, joint software reviews in Defence and where the development of systems is outsourced [DeMarco and Lister, 1998; Jones, 1998]. The second source of goal conflict arises from the objectives of the system and the objectives for its development and acquisition. The conflicting objectives of the system and its development are discussed first. Conflict in the goals of the participants can be related to conflict in the objectives of the system.

Conflicting objectives arise for even relatively simple systems. For example, Ulkucu describes the objectives for a decision aid - a Multiple Criteria Decision Making (MCDM) system and discusses the conflict between the objectives [Ulkucu, 1989].

There are several classes of system objectives, and conflict can occur within and between different classes of system objectives.

- There are financial objectives and conflicts. For example, it may not be possible to minimise both the cost of developing the system and the cost of maintaining the system [Ulkucu, 1989].
- There are design style considerations, and conflict between these considerations. For example, high levels of both modularity and granularity are desirable, but both cannot be maximised [Ulkucu, 1989].
- There are functional objectives. Conflicts in the functional objectives may arise from different expectations of different users.
- There are also non-functional requirements and conflicts can occur between these requirements. For example, a high response time and a high algorithm reliability might be desirable, but impossible to achieve [Ulkucu, 1989].
- There may also be conflicts between these different classes. For example producing a system with a high level of interactiveness may increase development costs, which are trying to be minimised [Ulkucu, 1989].

Trade-offs need to be made between these objectives at various times during the development. For example, as explained by Gabb below, trade-offs will be made during the detailed design phase. Gabb uses the term Project Authority (PA) to refer to the acquirers at the Defence project office, and the terms contractors and designers to refer to the developers.

> *"Detailed design will always result in "**design choices**", where the designers find it necessary to make choices which affect the functionality and performance of the system. In many cases these choices will be logical and acceptable to the PA. In others the designer will make a choice which is unacceptable to the users and/or acquirer. The need to make these choices will sometimes identify areas which are not specified or are underspecified. The designer cannot be expected to ask the PA each time he makes such a choice (there will be literally thousands of them in a medium scale project). The CDR gives the PA the opportunity to see the result of these choices and take appropriate action."* [Gabb, 1997]

The trade-offs chosen will depend on experience of the designers, the quality of the specification and the perceived benefits and limitations of alternative designs or development strategies.

Different individuals and groups or organisations will perceive different benefits and limitations to design alternatives. For example, as stated by Gabb, the best design from the developer's perspective will not always be the alternative preferred by the acquirer - and may even be unacceptable to the acquirer [Gabb, 1997].

This difference in perception relates to the second source of conflict - differences in the goals of the participants. In addition to the relative importance placed on the objectives of the system and its development, the participants may have other goals for a review. These goals may be in conflict with the goals of other participants.

For example, the author's main goal may be to have their product approved, or to minimise the amount of rework required [Gabb et al., 1992]. It is easy to see how this could conflict with the goals of other review participants.

3.4 Goal Conflict

This section discusses how goal conflict can affect the performance of software reviews. Section 3.3 discussed how goal conflict arises in software reviews, and Figure 11 introduced the idea that the goals of individuals, groups and organisations also affect the efficiency of reviews. (The individual, group, and organisational goals appear in the box on the left-hand side of the figure. The properties in this box influence both the group process and the group decision scheme, which in turn influence the performance of software reviews.)

Several combinations of goals are possible within a review and occur at two main levels - the group and the individual levels. The goals at each level may be either complementary or conflicting. Conflicting goals cannot all be met simultaneously. They require trade-offs to be made. Complementary goals can all be satisfied simultaneously. For example, a system may have goals of maximising the accuracy of a calculation, but minimising the time it takes for the calculation to be performed. These goals are conflicting and require trade-offs to be made. On the other hand, it may be possible to achieve both the goal of minimising the time it takes for the calculation and the goal of providing a user-interface which is easy to use. These goals are complementary. Conflicting goals may arise because the group and individuals have sets of conflicting goals, or because different individuals or subgroups have different goals, which conflict. The combinations of conflicting or complementary goals with the same or different subgroups are summarised in Table 2.

Table 2: Possible goal combinations

	Complementary		Conflicting	
Group	Same		Same	
Subgroup or Individual	Same	Different	Same	Different

Where sub-groups are present (e.g. when multiple organisations are present) the possible goal combinations are the same as those given for individuals within a group in Table 2.

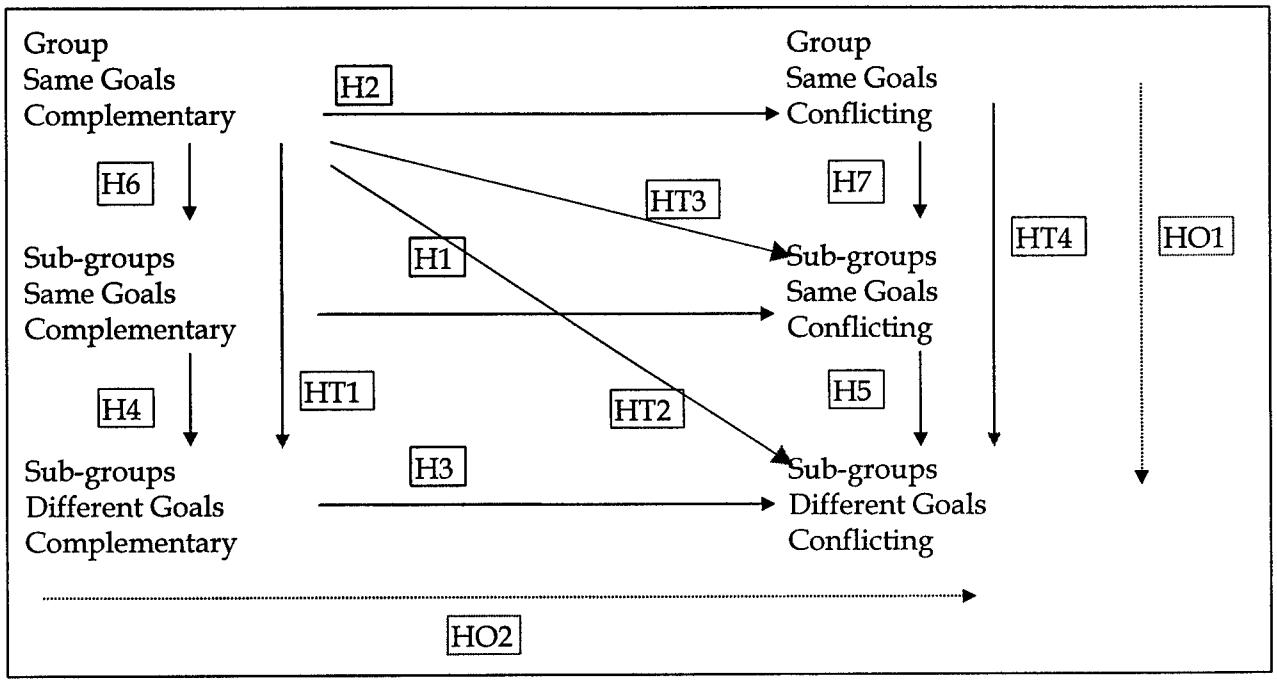

Figure 13: Summary of treatments and hypotheses

Several hypotheses are proposed in this section of the document. They are summarised in Figure 13. The lines correspond to hypotheses. The direction of the arrow is the direction of the expected performance *decreases*. Three types of labels are used on the lines: those starting HO are general, over-arching hypotheses; those starting with HT will follow if the other hypotheses are true, and the third set of hypotheses take the form H followed by a number. The hypotheses are numbered in the order that they are proposed in this section.

Studies from negotiation theory have shown that performance of group and individual activities is adversely affected by conflicting goals. For example, the number and quality of potential solutions raised during negotiations have been inversely linked to goal conflict [Nunamaker et al., 1991a; Foroughi et al., 1995]. (See Section 4.1 a description of how the efficiency of reviews may be determined using the number and quality of issues raised and resolved.)

It is hypothesised that the same will be true for software reviews:

H1. Software reviews by *partitioned sub-groups* with *different* goals will raise and resolve a *greater* number and quality of issues when the goals of the sub-groups are *complementary*, than when the goals of the sub-groups *conflict*.

Based on studies by Schater it has been argued that conflicting goals adversely affect the performance of all groups [Luthans, 1985]. For software reviews, we propose the additional hypotheses:

H2. Software reviews *without partitioned sub-groups* will raise and resolve a *greater* number and quality of issues when the goals of the group *are complementary,* than when the goals of the group are *conflicting.*

H3. Software reviews by *partitioned sub-groups* with the *same* goals will raise and resolve a *greater* number and quality of issues when the goals of the sub-groups are *complementary,* than when the goals of the sub-groups are *conflicting.*

And the general hypothesis:

HO2. Software reviews will raise and resolve a *greater* number and quality of issues when the goals of the group/participants are *complementary,* than when the goals of the group/participants are *conflicting.*

In the software engineering field, differences in the goals of stake-holders (e.g. developers, users, management) have been cited as a major cause of failure and termination of projects [Warne and Hart, 1995]. While Warne and Hart focused on the development of information systems, Jones describes how the Denver airport baggage handling system ending in litigation, like 5% of all outsourcing contracts [Jones, 1998]. Jones claims that Implicit Requirements - automating poorly understood manual process, ambiguous deliverables, and poorly defined quality criteria can cause conflict in projects which can ultimately end in litigation.

There is evidence of the impact of goals on software reviews. Most of the empirical studies on software reviews have focused on software inspections where the emphasis is on defect detection. One cause of inspection failure is the lack of, poorly defined, or differing inspection goals, standards, and quality goals [Brykczynski, 1994; Gilb, 1996; Grady and Van Slack, 1994; Shirey, 1992].

It is therefore hypothesised that the performance of reviews will depend on whether the group has a common set of goals, regardless of whether these goals are complementary or conflicting. The following two hypotheses are therefore proposed:

H4. Software reviews by *partitioned sub-groups* will raise and resolve a *greater* number and quality of issues when the goals of the sub-groups are the *same* and *complementary,* than when the goals of the sub-groups are *different* but *complementary.*

H5. Software reviews by *partitioned sub-groups* will raise and resolve a *greater* number and quality of issues when the goals of the sub-groups are the *same* but *conflicting,* than when the goals of the sub-groups are *different* and *conflicting.*

Well-run inspections take measures - such as restricting participation to peers, and not allowing reviewers to comment on design alternatives - to reduce the possibility of conflict e.g. [Freedman and Weinberg, 1982]. The effects of goal conflict on inspections

have not been hypothesised and indeed, it would be difficult to control goal conflict with the usual model of software inspections. It is easier to manipulate goal conflict within broader forms of software review.

Two additional hypotheses are generated from HO1 of Section 3.2 by considering its implications where the goals of the group are conforming and conflicting:

H6. Software reviews *without partitioned sub-groups* and with *conforming* goals will raise and resolve a *greater* number and quality of issues than reviews *with partitioned subgroups* and the *same, conforming* goals.

H7. Software reviews *without partitioned sub-groups* and *with conflicting goals* will raise and resolve a *greater* number and quality of issues than reviews *with partitioned subgroups* and the *same, conflicting* goals.

While hypotheses such as those above need to be tested to determine the effects of conflict on the performance of software reviews, several additional hypotheses can be derived by the transitive closure[7] of hypotheses H1 to H7. A third general hypothesis is derived by combining the hypotheses concerning goal differences:

HT1. Software reviews *without partitioned sub-groups* and with *complementary* goals will raise and resolve a *greater* number and quality of issues than software review *with partitioned sub-groups* with *different* but *complementary* goals.

HT2. Software reviews *without partitioned sub-groups* and with *complementary* goals will raise and resolve a *greater* number and quality of issues than software reviews with partitioned sub-groups with different and conflicting goals

HT3. Software reviews *without partitioned sub-groups* and with *complementary* goals will raise and resolve a *greater* number and quality of issues than software reviews *with partitioned sub-groups* with the *same,* but *conflicting* goals.

HT 4. Software reviews *without partitioned sub-groups* but with *conflicting* goals will raise and resolve a *greater* number and quality of issues than software reviews *with partitioned sub-groups* with *different* and *conflicting* goals.

HO3. Software reviews *with partitioned sub-groups* with the *same* goals will raise and resolve a *greater* number and quality of issues than reviews *with partitioned sub-groups* with *different* goals.

The next section discusses methods by which conflicting goals - both within a group and between subgroups - can arise.

---

[7] The transitive closure of a directed graph is a new directed graph with the same nodes as the original graph. The there is an arrow between two nodes in the new graph, if there was a path between them in the original graph. For example, the arrow HT1 in Figure 12 exists because there of several paths, including the path via H6, H4 and H3.

4. Implications and future work

The preceding sections of this document: 1) proposed a new model for software reviews, 2) introduced two new types of issues and 3) considered the impact of group structure and goal differences on software reviews. This section considers the implications of these concepts and how they can be further investigated. It looks at the effect of the two new types of issues on the measurement of review performance (Section 4.1) . Planned studies of the hypotheses are introduced in Section 4.2 and the final section discusses the implications for software reviews if these studies support the hypotheses.

4.1 Measuring the performance of reviews.

A measure of the performance of reviews is required to investigate the hypotheses proposed in this document. This section builds on the methods used for measuring the performance of reviews in laboratory experiments on inspections. This task is complicated by the fact that the number of issues present in a review product depends on the goals of the participants. This issue will be resolved in the early experiments by using the same sets of goals for each of the treatments under investigation - the only differences in the treatments will be due to the ways in which the goals are combined.

The efficiency of inspections is typically measured using a combination of the number of issues identified and the quality of the issues identified. However, inspections focus on one type of issues – defects. The issues identified in other software reviews take one of three forms - defects, risk mitigation (RM) and implicit requirements (IR). The RM and IR issues have many similarities and together are called goal-dependent issues. The measure of efficiency may need to consider defects and goal-dependent issues separately.

Three types of defects are commonly used in the literature on inspections: true defects, stylistic issues and false positives e.g. [Cheng and Jeffery, 1996a; Porter et al., 1995b]. Similarly, goal-dependent issues can be important, trivial or somewhere in between. We call this property the *quality* of the issues. The measure of efficiency for software reviews should also consider the quality of issues. This paper define 3 levels of quality for issues:

knowledge	issues which can be resolved by increasing the knowledge of the participants
stylistic	issues raised due to differences in formatting preferences etc. (these should not need to be resolved)
true	other types of issues

Note that knowledge issues are very similar to false positives. Knowledge issues are still considered to be issues rather than false positives because the knowledge transfer required to fix them is considered an important part of the review process.

Table 3: Factors in the Effectiveness of Software Reviews[8]

ISSUES	SCALE	FACTORS		
Defect	# present	# true defects	# false positives	# stylistic defects
RM Issues	# present	# true issues raised # true issues resolved	# knowledge issues raised # knowledge issues resolved	# stylistic issues
IR Issues	# present	# true issues raised # true issues resolved	# knowledge issues raised # knowledge issues resolved	# stylistic issues

Table 3 summarises the factors that affect the efficiency of software reviews. One measure of efficiency is a vector of these factors that has been normalised against some scale such as the number of issues present. In many cases, it will be possible to combine some of these factors to simplify the measure of efficiency. For example, depending on the circumstances there may be no need to distinguish between RM and IR issues, between issues and defects, or between true issues and knowledge issues. If the number of issues present can be controlled, then the number of issues raised and resolved need not be scaled.

Each of the hypotheses presented in Section 3 can be refined to consider the efficiency of software reviews in identifying a particular factor, or combination of factors from Table 3.

Example 3: Hypothesis Refinement
Consider the refinement of hypothesis H1. Firstly, assume we group the factors into two classes: issues raised and issues resolved. Then for hypothesis H1 we identify two new hypotheses:

H1-1: Software reviews with partitioned sub-groups with different goals will raise a greater number of issues when the goals of the sub-groups are complementary, than when the goals of the sub-groups conflict.

H1-2: Software reviews with partitioned sub-groups with different goals will resolve a greater number issues when the goals of the sub-groups are complementary, than when the goals of the sub-groups conflict.

Alternatively, we could use all of the factors and obtain thirteen new hypotheses including the following hypotheses on true issues:

H1-1: Software reviews with partitioned sub-groups with different goals will identify a greater number of true defects when the goals of the sub-groups are complementary, than when the goals of the sub-groups conflict.

H1-2: Software reviews with partitioned sub-groups with different goals will raise a greater number of true RM issues when the goals of the sub-groups are complementary, than when the goals of the sub-groups conflict.

[8] The symbol '#' is used in Table 3 as an abbreviation for 'number of'.

H1-3: Software reviews with partitioned sub-groups with different goals will resolve a greater number of true RM issues when the goals of the sub-groups are complementary, than when the goals of the sub-groups conflict.

H1-4: Software reviews with partitioned sub-groups with different goals will raise a greater number of true IR issues when the goals of the sub-groups are complementary, than when the goals of the sub-groups conflict.

H1-5: Software reviews with partitioned sub-groups with different goals will resolve a greater number of true RM issues when the goals of the sub-groups are complementary, than when the goals of the sub-groups conflict.

The methods described here are suitable for laboratory studies of software reviews. Additional work is required to develop measures of efficiency used for field studies. The most important consideration is how to determine a suitable scaling factor for the measure.

4.2 Empirical Studies

Empirical studies are planned to investigate the hypotheses proposed in this document. The early studies will be formal experiments conducted under laboratory conditions. These studies will require careful planning to ensure that the treatments are successfully applied: that artificially induced subgroups and goal conflict are created and maintained.

Goal conflict and group structure should be incorporated into the models of software reviews if these hypotheses presented here are supported (or not refuted) by the early studies. Additional studies would be required to further probe the effects of group structure and goal conflict. There are several avenues for additional study. The effects of alternative group structure could be explored. Methods could be identified for controlling the amount of goal conflict in reviews. A number of models of conflict resolution are available, each with some limitations [Lewicki et al., 1992] and it is not clear which model would be best suited to software reviews.) Interaction effects with factors that have previously been tested for inspections would also need to be investigated. For example, the effects of group size and group structure on review performance may not be independent. Finally, case studies need to be conducted to determine the effects of the theory outside of the laboratory setting.

These studies may not just benefit software engineering by increasing their knowledge about software reviews: studies on the effects of goal conflict may also contribute to our understanding of negotiation and conflict resolution. Empirical studies in these areas have tended to focus on business applications e.g. [Nunamaker et al., 1991b]. Studies in software reviews would add support for the general acceptance of any negotiation and conflict resolution principles investigated.

4.3 Implications

Three general hypotheses HO1-HO3 were proposed in Section 3. This section discusses some possible modification to the joint software review processes, which may be desirable in light of each hypothesis.

Subgroups with different goals
Hypothesis HO3 proposes that differences in the goals of sub-groups adversely affects the performance of reviews. The author can identify two possible mechanisms for reducing the differences between the goals of the organisations present at a review. The first requires a small change - adding a goal identification stage to the review. The second requires a change to the way in which contracts are implemented.

Adding a goal-identification stage to joint software review processes may improve their efficiency if hypothesis HO3 holds. This stage could be used to combine the goals of the developer and the acquirer and identify a common set of goals for the remainder of the review. This stage could be conducted at the start of the review or when the review products are distributed for review. It may be possible to identify some review goals before contract negotiation. The benefits of incorporating this stage into the review process would need to be weighed against the costs before deciding whether or not to modify the review process. For example, the effort and time required to determine common goals for the review may outweigh any savings made during the remainder of the review process.

A second method for reducing the number of differences in the goals of the developers and contractors would be to have a combined acquirer/contractor development team. This has been proposed as a method for improving the acquisition process in general (e.g. [Henderson and Gabb, 1997]). While there was no previous theoretical basis for this recommendation, it was proposed as it was felt that a partnership would be more efficient that the current, often adversarial relationship. The hypotheses presented in this paper do not address the entire acquisition process, so only the impact on software reviews is considered here. Creating a cohesive team from members of both the acquirer and the contractor would address two of the hypotheses considered in this paper. Firstly, a single set of review goals would be more likely (HO3) and secondly the review members would come from a single, more cohesive team (HO1). It is not clear whether it would also reduce the level of conflict within the group's goals (HO2).

Subgroups
In the author's opinion, it is not likely that other methods of obtaining a single, cohesive group (HO1) for the review would be particularly effective, as they would require people to act as a single group for the review, but would be required to act independently at other times.

However, other changes to the review process which mask the distinction of the two sub-groups (the acquirers and the contractors) may be beneficial. For example the following may be beneficial: seating arrangements where contractors and developers are interspersed; the use of groupware for eliciting comments (e.g. [Nunamaker et al., 1991a]); the introduction of additional group or team-building activities where all the participants must work together (e.g. [Henderson and Gabb, 1997]), the use of

integrated product teams (e.g. [Sterzinger, 1998]); or the exchange of personnel between the two groups or organisations.

Conflicting Goals
Hypothesis HO2 proposes that conflict in the group's goals adversely affects the performance of reviews. The author can identify two possible mechanisms for reducing goal conflict. As in Subgroups with Different Goals, the first mechanism requires a small change to the review process, and the second requires a change to the way in which contracts are implemented.

The first mechanism is an extension of the goal identification phase described in the first part of this section. Instead of just deciding on a common set of goals, the goals are negotiated, and conflicting goals are weighted and prioritised. Agreement on the priorities and goals would enable conflicts to be discussed in a rational manner.

A second possible method of addressing the level of conflict within the group's goals is at the contractual stage. The use of incentive schemes, which reward the contractors for acting in the interests of the acquirers, can significantly reduce the conflict in the goals of the two parties. (This approach has been used in other fields.)

One project where this type of reward scheme was used is the ANZAC ship project. By including a maintenance period in the initial contract, the developers have an incentive to minimise the whole life-cycle costs for the system, and not just the development costs. However, care needs to be taken to ensure that this approach does not restrict the possible system maintainer after the completion of the contract.

A second project that has attempted this in a different manner is the ADFDIS project where the contractors agree to follow a particular development process rather than develop a system according to detailed system requirements. The development process they agreed to includes identification of user requirements, continual user feedback and implementation of an improvement cycle where these changes are fed back into the development.

Care needs to be taken in developing incentive schemes: to ensure that the scheme does indeed provide incentive for the contractors to act in the government's best interest, and to ensure that the schemes do not restrict the options of Defence in the long term.

5. Conclusions

It has been argued that software inspections are "better" than other forms of software review e.g. [Ackerman et al., 1989; Britcher, 1988]. However, it is not sufficient to rely solely on software inspections to identify issues under all circumstances. For example, inspections and joint software reviews may be required when the developers and the users come from different organisations, as in most Defence acquisitions. This paper identifies some important limitations with current models of software reviews due to their focus on inspections. These limitations highlight the importance of studying the broader field of software reviews rather than focusing solely on software inspections.

This paper proposes a model on software reviews and identifies two new areas for research into software reviews. Several hypotheses are drawn in the new areas and they are currently being evaluated using experimental techniques. If these experiments support the theory then the new areas form an important field of study for all forms of software review. Through their application to inspections they are important for the study of systems development and, perhaps more importantly, through their application to joint reviews, they are important for the study of software acquisition. The new areas identified may be harder to study than areas that have been previously studied for inspections. The new areas require more structure and control than many of the other areas studied. However, this does not mean that group structure and goal conflict should not be investigated further. For example, if the presence of sub-groups or the presence of observers severely reduces the performance of reviews then measures should be taken to limit the impact. The author is currently investigating some of these areas further and will publish a series of reports on software reviews.

Several possible improvements to the joint software review process, and to the broader acquisition process, have been identified on the basis of the hypotheses proposed in this paper. Some of these suggestions require significant changes to the way in which software-intensive systems are currently acquired. Additional supporting evidence should be obtained before they are introduced to Defence, and to industry, on a wider scale. This evidence may include anecdotal evidence, but the prime source of evidence should be formal experiments and case studies. In capturing additional evidence the application of these ideas to other domains: such as the review of systems should also be considered. While this report draws on the software inspection literature and does not explicitly address the review of systems, significant differences between software reviews and systems reviews are not expected.

Acknowledgments

Thanks are due to all the people who reviewed this report, participated in discussions and reviewed related work. Peter Fisher and Stefan Landherr of the Year 2000 Cell (formerly in the SSE Group) reviewed early drafts. Stefan Landherr also participated in several discussions on software reviews, as did Prof Ross Jeffery at the University of New South Wales where the author is undertaking a PhD. Prof Jeffery also reviewed material related to this document. Alex Yates and Dr Rudi Vernik of the Software Systems Engineering Group reviewed later drafts.

References

[Ackerman, A F et al., 1989] "Software Inspections: An Effective Verification Process." *IEEE Software* 6(3):pp 31-36.

[Ackerman, A F et al., 1984] Software Inspections and the Industrial Production of Software. Software Validation. Amsterdam, Elsevier. pp13-40.

[ADO, 1996] Defence Project Management: Pitfalls and Pointers (Proceedings). Canberra. **Australian Defence Studies Centre**.

[Anderson, K J et al., 1988] "Reuse of Software Modules." *AT&T Technical Journal* Jul/Aug: pp 71-76.

[ANSI/IEEE-1028, 1989] IEEE Standard for Software Reviews and Audits (No. IEEE).

[Baldwin, J T, 1992] An Abbreviated C++ Code Inspection Checklist, University of Illinois, Department of Computer Science.

[Barnard, J and Price, A, 1994] "Managing Code Inspection Information." *IEEE Software* 11(2):pp 59-69.

[Bisant, D B and Lyle, J R, 1989] "A Two-Person Inspection Method to Improve Programming Productivity." *IEEE Transactions on Software Engineering* 15(10):pp 1294-1304.

[Britcher, R N, 1988] "Using Inspections to Investigate Program Correctness." *IEEE Computer* 21(11):pp 38-44.

[Brothers, L R, 1992] "Multimedia groupware for code inspection". *SUPERCOMM/ICC '92. Discovering a New World of Communications*, Chicago, IL, USA. IEEE.

[Brykczynski, B, 1994] Why Isn't Inspection Used, Posting to comp.software-eng, 16 Mar 1994, 12:07:34 GMT, .

[Buck, F O, 1981] Indicators of Quality Inspections, Technical Report (No. TR21.802), IBM Corporation.

[Canale, R and Wills, S, 1995] "Producing professional interactive multimedia: project management issues." *British Journal of Educational Technology* 26(2):pp 84-93.

[CEPMAN 1, 1996] The Capital Equipment Procurement Manual AL12 (No. Revision AL12 (8/8/96)), Acquisition and Logistics, Australian Department of Defence.

[Charette, R N, 1994] Risk Management. Encyclopedia of Software Engineering. New York, John Wiley & Sons, Inc. pp1091-1106.

[Cheng, B and Jeffery, R, 1996a] Apparatus for an Experiment of Function Point Scenarios (FPS) for Software Requirement Specification, Technical Report (No. 96/4), Centre for Advanced Empirical Software Research, University of New South Wales.

[Cheng, B and Jeffery, R, 1996b] "Function Point Scenarios (FPS) for Software Requirement Specification." *DRAFT*.

[Chung, W Y and Guinan, P J, 1994] "Effects of Participative Management on the Performance of Software Development Teams".

[Davis, K and Newstrom, J W, 1985] Human Behavior at Work: Organisational Behavior. McGraw-Hill.

[DeMarco, T and Lister, T, 1998] "Both Sides Always Lose: The Litigation of Software-Intensive Contracts." *Cutter IT Journal* 11(4):pp 5-9.

[DI(N)LOG 10-3, 1995] Operational Software Engineering Management (OSEM) (No. Refer DSTO file N8319/3/6. Copy appears on Defence Managers' Toolbox CD-ROM).

[DOD-STD-2167A, 1988] DOD-STD-2167A Defense System Software Development (No. DOD-STD-2167A).

[Earnshaw, A A P, 1994] The Acquisition of Major Capital Equipment by the Australian Department of Defence: A Comparative Analysis, PhD, University of Canberra.

[Fagan, M E, 1976] "Design and Code Inspections to Reduce Errors in Program Development." *IBM Systems Journal* 15(3):pp 182-211.

[Fagan, M E, 1986] "Advances In Software Inspections." *IEEE Transactions on Software Engineering* 12(7):pp 744-751.

[Fisher, P D et al., 1997] Lessons Learned from Software Engineering Support to JORN, Client Report (Limited Release) (No. DSTO-CR-0050), DSTO.

[Foroughi, A et al., 1995] "An Empirical Study of an Interactive, Session-Oriented Computerized Negotiation Support System (NSS)." *Group Decision and Negotiation* 4:pp 485-512.

[Fowler, P J, 1986] "In-Process Inspections of Workproducts at AT&T." *AT&T Technical Journal* 65(2):pp 102-112.

[Freedman, D P and Weinberg, G M, 1982] Handbook of Walkthroughs, Inspections, and Technical Reviews: Evaluating Programs, Projects, and Products. Mass. Boston, Little, Brown & Co.

[Gabb, A, 1997] Critical Design Reviews - A Pragmatic Approach, Informal Document, 23 July 1997, .

[Gabb, A P et al., 1991] Tailoring DOD-STD-2167A - A Survey of Current Usage, Technical Note (No. WSRL-TN-57/91), DSTO.

[Gabb, A P et al., 1992] Recommendations for the Use and Tailoring of DOD-STD-2167A, Research Report (No. ERL-0637-RE), DSTO.

[Gilb, T, 1996] "Inspection Failure Causes." *Testing Techniques Newsletter* October(On-line Edition).

[Grady, R B and Van Slack, T, 1994] "Key Lessons in Achieving Widespread Inspection Use." *IEEE Software* 11(4):pp 46-57.

[Haimes, Y, Chittister, C., 1993] An acquisition Process for the Management of Risks of Cost Overrun and Time Delay Associated with Software Development, Technical Report (No. CMU/SEI-93-TR-28), Software Engineering Institute.

[Heemstra, F J, 1992] "Software Cost Estimation." *Information and Software Technology* 34(10):pp 627-639.

[Henderson, D and Gabb, A, 1997] Using Evolutionary Acquisition for the Procurement of Complex Systems, Technical Report (No. DSTO-TR-0481), DSTO.

[Humphrey, W S, 1995] A Discipline for Software Engineering. Addison-Wesley Publishing Company.

[ISO/IEC 9126-1, 1996] Information Technology - Software quality characteristics and metrics - **Part 1: Quality characteristics and sub-characteristics**, Standard (No. ISO/IEC 9126-1).

[ISO/IEC 12207, 1995] Information Technology Software Life Cycle Processes, Standard (No. ISO/IEC 12207), ISO/IEC.

[Jones, C, 1998] "Conflict and Litigation Between Software Clients and Developers." *Cutter IT Journal* 11(4):pp 10-20.

[Keil, M, 1995] "Identifying and Preventing Runaway Systems Projects." *American Programmer*(March):pp 16-22.

[Kim, L P W et al., 1995] A Framework for Software Development Technical Reviews. Software Qualtiy and Productivity: Theory, practice, education and training., Chapman & Hall. pp294-299.

[Knight, J C and Myers, E A, 1993] "An Improved Inspection Technique." *Communications of the ACM* 36(11):pp 51-61.

[Lanubile, F and Visaggio, G, 1995] "Assessing Defect Detection Methods for Software Requirements Inspections Through External Replication." *Emperical Software Engineering*.

[Lederer, A L and Prasad, J, 1995] "Causes of inaccurate software development cost estimates." *Journal of Systems and Software* 31(2):pp 125-34.

[Letovsky, S, et al, 1987] A Cognitive Analysis of a Code Inspection. Empirical Studies of Programming. Norwood, N.J., Ablex. pp231-247.

[Lewicki, R J et al., 1992] "Models of conflict, negotiation and third party intervention: A review and synthesis." *Journal of Organizational Behaviour* 13:pp 209-252.

[Luthans, F, 1985] Organizational Behaviour. McGraw-Hill.

[Macdonald, F et al., 1995] "A Review of Tool Support for Software Inspection". *7th International Workshop Computer-Aided Software Engineering*, Los Alamitos, CA. IEEE CS Press.

[Marciniak, J J , Ed. 1994] Encyclopedia of Software Engineering, John Wiley & Sons, Inc.

[Martin, J and Tsai, W-T, 1990] "N-fold Inspection: A Requirements Analysis Technique." *Communications of the ACM* 33(2):pp 225-232.

[McCarthy, P et al., 1996] An Experiment to Assess Cost-Benefits of Inspection Meetings and their Alternatives: A Pilot Study.

[MIL-STD-498, 1994] Software Development and Documentation, AMSC (No. N7069), Department of Defense (USA).

[MIL-STD-498, 1996] MIL-STD-498 Overview and Tailoring Guidebook, Department of Defense (USA).

[MIL-STD-1521B, 1985] Military Standard for Technical Reviews and Audits for Systems, Equipments, and Computer Software, Military Standard (No. MIL-STD-1521B), US Dept. of Defense.

[Mosemann II, L K, 1995] "Software Development: Quo Vadis?" *CrossTalk: The Journal of Defense Software Engineering* 8(11):pp 2-3.

[Myers, E A and Knight, J C, 1992] An Improved Software Inspection Techniques and an Empirical Evaluation of Its Effectiveness, Technical Report (No. CS-92-15), University of Virginia, Department of Computer Science.

[Nielsen, R P, 1979] "Stages in moving toward cooperative problem solving labor relations projects and a case study." *Human Resource Management* Fall:pp 2-8.

[Nunamaker, J F et al., 1991a] "Information Technology for Negotiating Groups: Generating Options for Mutual Gain." *Management Science* 37(10):pp 1325-1346.

[Nunamaker, J F et al., 1991b] "Electronic Meeting Systems to Support Group Work." *Communications of the ACM* 34(7):pp 42-61.

[Parnas, D L and Weiss, D M, 1985] "Active Design Reviews: Principles and Practices". *8th International Conference on Software Engineering*, London, England.

[Parnas, D L and Weiss, D M, 1987] "Active Design Reviews: Principles and Practices." *Journal of Systems and Software* 7:pp 259-265.

[Porter, A A et al., 1995a] "Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment." *IEEE Transactions on Software Engineering* 21(6):pp 563-575.

[Porter, A A et al., 1995b] "An Experiment to Assess the Cost-Benefits of Code Inspections in Large Scale Software Development." *SIGSOFT Software Engineering Notes* 20(4):pp 92-103.

[Rescher, N, 1983] Risk. University Press of America.

[Remus, H, 1984] Integrated Software Validation in the View of Inspections/Reviews. Software Validation. Amsterdam, Elsevier. pp57-64.

[Rifkin, S and Deimel, L, 1994] "Applying Program Comprehension Techniques to Improve Software Inspections". *19th Annual NASA Software Engineering Laboratory Workshop.*

[Sauer, C et al., 1996] "A Behaviourally Motivated Programme for Empirical Research into Software Development Technical Reviews." *Technical Report (No 96/5) CAESAR, University of New South Wales.*

[Schneider, G M et al., 1992] "An Experimental Study of Fault Detection in User Requirements Documents." *ACM Transactions on Software Engineering and Methodology* 1(2):pp 188-204.

[Shirey, G C, 1992] "How Inspections Fail". *9th International Conf. Testing Computer Software.*

[Simmons, D B, 1991] "Communications: A software group productivity dominator." *Software Engineering Journal* 6(6):pp 454-462.

[Sterzinger, R F, 1998] Integrated Product Team (IPT) Guide, The Boeing Company.

[Tripp, L L et al., 1991] "The Application of Multiple Team Inspections on a Safety-Critical Software Standard". *4th Software Eng. Standards Application Workshop.* IEEE CS Press.

[Tversky, A and Kahneman, D, 1989] Rational Choice and the Framing of Decisions. Multiple Criteria Decision Making and Risk Analysis Using Microcomputers, Springer-Verlag. pp81-126.

[Ulkucu, A, 1989] Conflicting Objectives in Software System Design. Multiple Criteria Decision Making and Risk Analysis Using Microcomputers, Springer-Verlag. pp357-394.

[VanHee, L, 1996] Software Quality Assurance, A Risk Management Approach. http://www.deepharbor.com/sqa/risk.html.

[Votta, L G, Jr., 1993] "Does Every Inspection Need a Meeting?" *1st ACM SIGSOFT Symposium on the Foundations of Software Engineering* 18(5):pp 107-114.

[Walsh, M, 1994] "Why software continues to cost more-and what the IS auditor can do." *EDPACS* 21(11):pp 8-15.

[Warne, L and Hart, D, 1995] "Organizational Power and Information Systems Development: Findings from a Case Study of a Large Public Sector Project". *6th Australasian Conference on Information Systems.*

[Weinberg, G M and Freedman, D P, 1984] "Reviews, Walkthroughs, and Inspections." *IEEE Transactions on Software Engineering* SE-10(1):pp 68-72.

[Weller, E F, 1993] "Lessons from Three Years of Inspection Data." *IEEE Software* 10(5):pp 38-45.

[Wheeler, D A et al., 1996] Software Inspection: An Industry Best Practice. Los Alamitos, CA, USA. IEEE Computer Society Press.

A Model for Joint Software Reviews

Gina Kingston

(DSTO-TR-0735)

DISTRIBUTION LIST

	Number of Copies
AUSTRALIA	
DEFENCE ORGANISATION	
S&T Program	
Chief Defence Scientist)	
FAS Science Policy)	1 shared copy
AS Science Corporate Management)	
Director General Science Policy Development	1
Counsellor, Defence Science, London	Doc Control Sheet
Counsellor, Defence Science, Washington	Doc Control Sheet
Scientific Adviser to MRDC Thailand	Doc Control Sheet
Director General Scientific Advisers and Trials)	1 shared copy
Scientific Adviser - Policy and Command)	
Navy Scientific Adviser	1 copy of Doc Control Sheet and 1 distribution list
Scientific Adviser - Army	Doc Control Sheet and 1 distribution list
Air Force Scientific Adviser	1
Director Trials	1
Aeronautical & Maritime Research Laboratory	
Director	1
Electronics and Surveillance Research Laboratory	
Director	1
Chief Information Technology Division	1
Research Leader Command & Control and Intelligence Systems	1
Research Leader Military Computing Systems	1
Research Leader Command, Control and Communications	1
Executive Officer, Information Technology Division	Doc Control Sheet
Head, Information Warfare Studies Group	Doc Control Sheet
Head, Software Systems Engineering Group	1
Head, Year 2000 Project	Doc Control Sheet
Head, Trusted Computer Systems Group	Doc Control Sheet
Head, Advanced Computer Capabilities Group	Doc Control Sheet
Head, Systems Simulation and Assessment Group	Doc Control Sheet
Head, C3I Operational Analysis Group	Doc Control Sheet
Head Information Management and Fusion Group	1
Head, Human Systems Integration Group	Doc Control Sheet

Head, C2 Australian Theatre	1
Head, Information Architectures Group	1
Head, Distributed Systems Group	Doc Control Sheet
Head C3I Systems Concepts Group	1
Head, Organisational Change Group	Doc Control Sheet
Author (Gina Kingston)	1
Publications and Publicity Officer, ITD	1

DSTO Library and Archives

Library Fishermens Bend	1
Library Maribyrnong	1
Library Salisbury	2
Australian Archives	1
Library, MOD, Pyrmont	Doc Control Sheet

Capability Development Division

Director General Maritime Development	Doc Control Sheet
Director General Land Development	Doc Control Sheet
Director General C3I Development	Doc Control Sheet

Army

ABCA Office, G-1-34, Russell Offices, Canberra	4
NAPOC QWG Engineer NBCD c/- DENGRS-A, HQ Engineer Centre	

Intelligence Program

DGSTA Defence Intelligence Organisation	1

Acquisition Program

DGCOMMS	1
DGCSS	1

Corporate Support Program (libraries)

OIC TRS Defence Regional Library, Canberra	1
Officer in Charge, Document Exchange Centre (DEC),	Doc Control Sheet & Distribution List
US Defence Technical Information Center	2
UK Defence Research Information Centre,	2
Canada Defence Scientific Information Service,	1
NZ Defence Information Centre,	1
National Library of Australia,	1

Universities and Colleges

Australian Defence Force Academy	1
Library	1
Head of Aerospace and Mechanical Engineering	1
Deakin University, Serials Section (M list), Deakin University Library,Geelong, 3217	1
Senior Librarian, Hargrave Library, Monash University	1
Librarian, Flinders University	1

Other Organisations

NASA (Canberra)	1
AGPS	1
State Library of South Australia	1
Parliamentary Library, South Australia	1

OUTSIDE AUSTRALIA

Abstracting and Information Organisations

INSPEC: Acquisitions Section Institution of Electrical Engineers	1
Library, Chemical Abstracts Reference Service	1
Engineering Societies Library, US	1
Materials Information, Cambridge Scientific Abstracts	1
Documents Librarian, The Center for Research Libraries, US	1

Information Exchange Agreement Partners

Acquisitions Unit, Science Reference and Information Service, UK	1
Library - Exchange Desk, National Institute of Standards andTechnology, US	1

SPARES 10

Total number of copies: 66

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
DOCUMENT CONTROL DATA

1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)

2. TITLE

A Model for Joint Software Reviews

3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION)

Document	(U)
Title	(U)
Abstract	(U)

4. AUTHOR(S)

Gina Kingston

5. CORPORATE AUTHOR

Electronics and Surveillance Research Laboratory
PO Box 1500
Salisbury SA 5108

6a. DSTO NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DSTO-TR-0735	AR-010-661	Technical Report	October 1998

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. NO. OF PAGES	12. NO. OF REFERENCES
N8316/7/34	97/243	DST	44	83

13. DOWNGRADING/DELIMITING INSTRUCTIONS

N/A

14. RELEASE AUTHORITY

Chief, Information Technology Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE CENTRE, DIS NETWORK OFFICE, DEPT OF DEFENCE, CAMPBELL PARK OFFICES, CANBERRA ACT 2600

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CASUAL ANNOUNCEMENT Yes

18. DEFTEST DESCRIPTORS

Computer programs
Reviews
Models

19. ABSTRACT

Joint software reviews, involving the developer and the acquirer, play an important role in Defence's acquisition of software-intensive systems. However, academic and commerical work on software reviews has focused on intra-organisational peer reviews and software inspections. This report argues that the principles which have been derived for inspections cannot be blindly applied to joint software reviews.

This paper proposes a model of joint reviews, which draws on software engineering, decision and negotiation theory, and models of inspection. The model suggests that the structure and goals of the review group may significantly affect the outcome of the review. The model has also been used to suggest changes to Defence's software review process and to plan a series of studies on joint software reviews. These studies will provide additional and updated recommendations on how Defence should structure their software reviews for maximum efficiency and effectiveness.